

TDB-ACC-NO: NN9311615

DISCLOSURE TITLE: Graphical Query System

PUBLICATION-DATA: IBM Technical Disclosure Bulletin,
November 1993, US

VOLUME NUMBER: 36

ISSUE NUMBER: 11

PAGE NUMBER: 615 - 616

PUBLICATION-DATE: November 1, 1993 (19931101)

CROSS REFERENCE: 0018-8689-36-11-615

DISCLOSURE TEXT:

This document contains drawings, formulas, and/or symbols that will not appear on line. Request hardcopy from ITIRC for complete article.

- Sometimes a chip designer will want to know: do I have an inverter connected to an inverter? Or there may be a replacement operation: find an NOR connected to a NOT, and replace them with an OR. This is not a global change NOR to OR, it requires examining the connectivity and perhaps other information. The query method can also be used in an exploratory fashion, complementing the interface provided by a browser/editor.

- Current tools used to modify an intermediate design are editors and logic synthesis. Editors have graphical interfaces that use the net/line, block/box metaphor. In general this interface is not suited to making complex global changes. With current technology it has become hard to present global chip design data in a

meaningful

way. Logic synthesis systems are powerful but generally not

interactive. Because the building blocks of a synthesis system are compiled programs, synthesis is not well suited for "one shot" problems.

- Graphical Query System (GQS) queries and modified VLSI design data by specifying configurations. This VLSI data consists of connectivity information with attributes, such as block function or power level. Configurations are patterns of connected elements, with partially filled in attributes. For example, a simple configuration might consist of an AND-function block connected to an INVERTER-function. There might be a large number of instances of this configuration in a design. Configurations are globally applied by GQS against the data.

- The interface to GQS is entirely graphical, following the traditional box and line representation for ECAD data. Useful

configurations may be very complex. A graphical notation was chosen

to make the tool's interface intuitive.

- The user is presented with an interface like that of a simple

schematic capture editor. These are two drawing windows, one for the query pattern and one for the replace pattern (if a modification is

desired). Buttons ^(1a) select the type of object and window to activate

(query/replace). ^(1b) Using the mouse, blocks and wires are drawn in the

appropriate window. Connectivity in the pattern is determined by

placement. Free wire ends are the i/o's to the pattern. All the

circuits are represented by generic boxes. Pressing another button

causes solutions to be found or displayed.

- Handling replacement requires a way to specify how the replacing pattern will be stitched into the design. In the query window dangling nets are the IOs to the pattern, for each such net a pseudo IO block is displayed in the replacement window. Connecting a net to an IO block in the replacement window creates an identity between this net and the dangling net in the query pattern.
- Here block C will replace the connected blocks A and B (as well as net N1). The dangling net, N2, caused the io box IO to be created. And the replace block, C, will have it's output N3 be the same net as N2 in the query pattern. These are really variable names, there may be many instances of this pattern in the design.
- The user interface must also provide an easy way to specify attributes. A button press on an object brings up an attribute window that is color coded to match the object it is associated with. Attribute values, such as block function or block power level are all optional, but may be specified through this window.
- Query ordering is based on user input and is vital to managing query complexity. It is most important that the search start with the most constrained elements. The algorithm used is to start the search with the first block placed by the user, then following connectivity, first source to sink, then sink to source. This is a compromise between requiring the user to determine the full query ordering and requiring the program to determine the best query ordering.
- The preliminary work was written using Quintus* Prolog on an

IBM RISC System/6000**, the current system has been rewritten in C++

and also runs on an RISC System/6000.

- Performance is relative to the size of the design and the number of answers. It is possible to specify queries with large

search spaces (any box connected to any box...). The system is

reasonably fast for well constrained queries. Finding 1K answers in

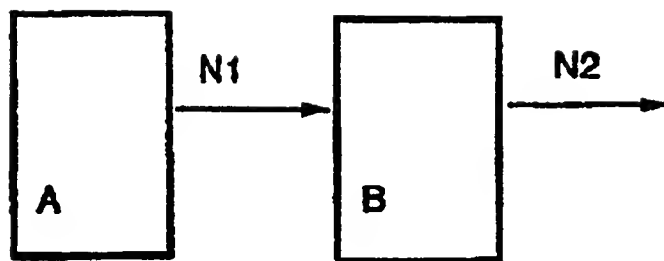
a design of 8K blocks takes about 5 seconds.

* Trademark of Quintus Corporation

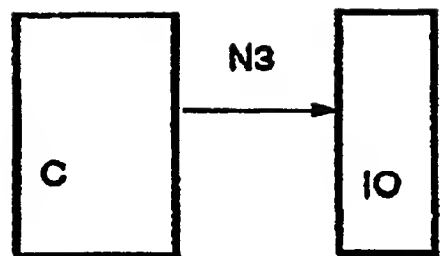
** Trademark IBM Corp.

SECURITY: Use, copying and distribution of this data is subject to the restrictions in the Agreement For IBM TDB Database and Related Computer Databases. Unpublished - all rights reserved under the Copyright Laws of the United States. Contains confidential commercial information of IBM exempt from FOIA disclosure per 5 U.S.C. 552(b)(4) and protected under the Trade Secrets Act, 18 U.S.C. 1905.

COPYRIGHT STATEMENT: The text of this article is Copyrighted (c) IBM Corporation 1993. All rights reserved.



Query window



Replace window